Assignment 2

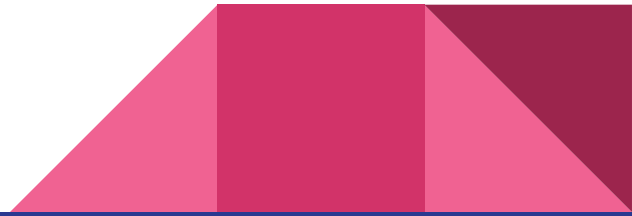
# Queuing System Simulation

Team Members:

- Ajinkya Tanksale (213050034)
- Arnav Mishra (213059002)

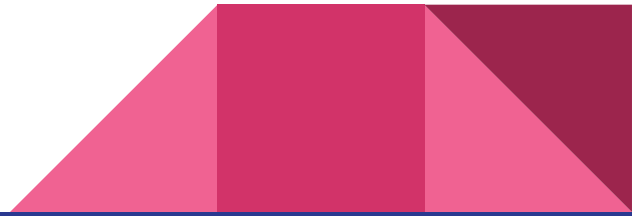
# Outline

1. System Assumptions
2. Constants User inputs and Enums
3. Classes
4. Code Highlight
5. System Architecture
6. Simulation Experiments and Results
    a. Comparison with Measurement data
    b. Multithreaded web Server Simulation
    c. Curiosity Experiment 1: Decreased Timeout Value
    d. Curiosity Experiment 2: Context Switch Time Variation
7. Conclusion



# System Assumptions

- System Type - Closed System
- Each user issues a fixed number of requests.
- Number of cores: 4
- Number of Maximum Threads per core: 4
- Request Buffer Size: 500
- On time-out, requests are retried. There is no limit for retries.
- Requests are dropped only if buffer is empty. User retries for them after timeout.
- Thread-to-Core Affinity
- Thread per request model



# Constants User inputs and Enums

**Constants**-
- Max_buffer_Size
- Max_thread_count
- Conext_switch_time
- Max_Request_Generated

**Users Inputs** –
- Mean_interarrival,
- Mean_service
- Number_of_users

**Scheduling Policy** ***(Enum)***
- FCFS (1)
- Round Robin (2)

**Server Status** ***(Enum)***
- Idle (1)
- Busy (2)

**Event Types** ***(Enum)***
- Arrival (1)
- Departure (2)
- Context_Switch_In (3)

**Distribution Type** ***(Enum)***
- Exponential (1)
- Uniform (2)
- Constant (3)

# Classes

**Service_time**

- *Attributes:*
  - typeOfDistribution(Enum Distribution type)
- *Methods:*
  - getServiceTime() /*Generation function{describes the distribution}*/

**Timeout**

- *Attributes:*
  - constantTime (double)
  - typeOfDistribution(Enum Distribution type)
- *Methods:*
  - getTimeoutTime() /*Generation function{describes the distribution}*/

# Classes

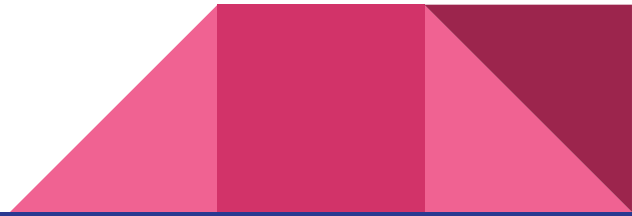
**Event**

- Attributes:
  - arrival_time (double)
  - timeout(double)
  - serviceTime
  - core (int)
  - thread(int)
  - response_count
- Methods:
  - getRandomThinkTime()/*random value chosen in range [4,10]*/
  - getRemainingServiceTime()

# Classes

## Core

- *Attributes:*
  - threads [Max Thread Count] (Event Object List)
  - status (int) {Server Status}
  - thread_busy_count (int)
- *Methods*:
  - GetCoreStatus()
  - setCoreStatus()
  - addToThread()
  - removeFromThread()
  - getBusyThreadCount()
  - setBusyThreadCount()



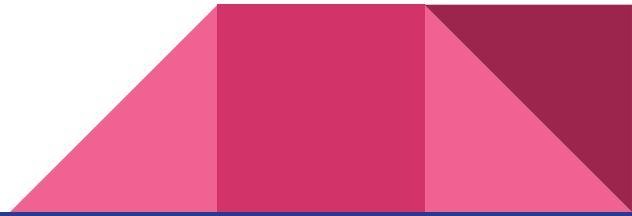
# Classes

**Scheduler**

- *Attributes:*
  - Type (int) {Scheduling policy}
  - Context_switch_time (double)

**Server**

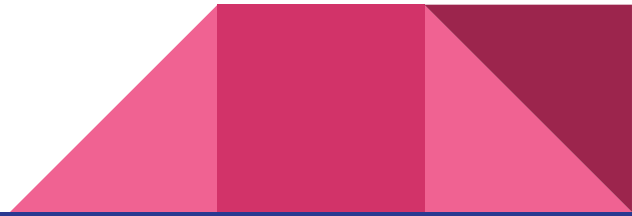
- *Attributes:*
  - Core Object [4];
  - service_time Object;
  - Scheduler Object;
  - {Waiting Buffer} Event Obj queue [Max buffer size]  (shared among all cores)
- *Methods:*
  - getNextEventFromBuffer()
  - getServerStatus()
  - setServerStatus()
  - getCoreObj()



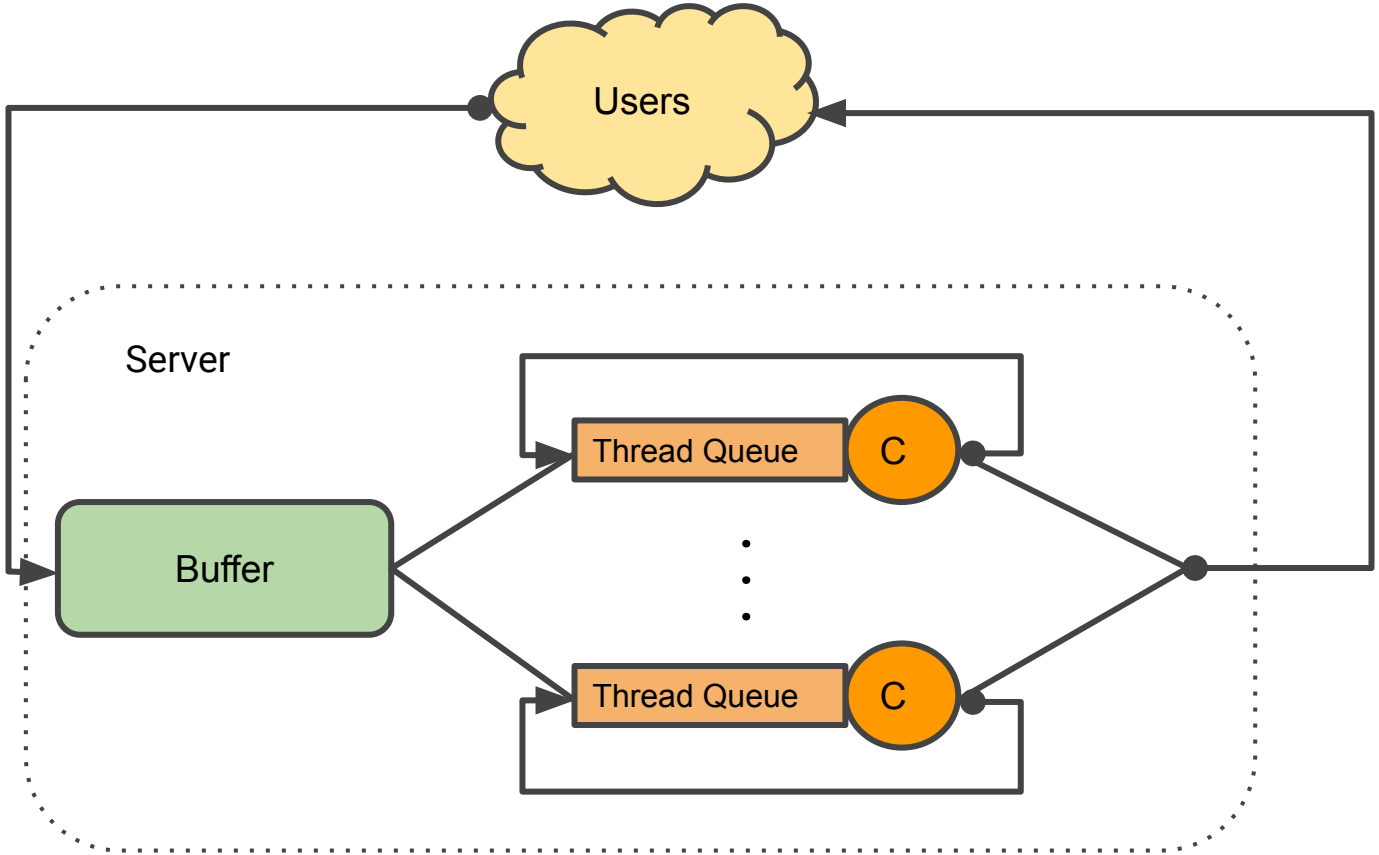
# Classes

**Event Handler**

- *Attributes*:
  - Server Obj
  - timing_next_event[Max_event_count] (a  priority queue of tuples <event_time, event obj> prioritized on event_time)
  - Timeout Obj
- *Methods*:
  - getNextEvent()
  - manageEvent()
  - Arrive()
  - Depart()
  - getServerObj()
  - setEvent()



# System Architecture


Users
Server
Thread Queue
C
Buffer
Thread Queue
C

# Code Highlights

# Code Highlights

## Server Log Output

Trace.txt

```
88.3154    [A A A I ]  EMPTY    Cntx Swtch In     88.4154
==========================================================================
88.4154    [A A A I ]  EMPTY    Departure         88.4806
==========================================================================
88.4806    [A I A I ]  EMPTY    Arrival           88.7719
==========================================================================
88.7719    [A A A I ]  EMPTY    Departure         88.787
==========================================================================
88.787     [I A A I ]  EMPTY    Cntx Swtch In     88.8719
==========================================================================
88.8719    [I A A I ]  EMPTY    Cntx Swtch Out    89.3719
==========================================================================
89.3719    [I A A I ]  EMPTY    Cntx Swtch In     89.4719
==========================================================================
89.4719    [I A A I ]  EMPTY    Departure         89.7606
==========================================================================
89.7606    [I A I I ]  EMPTY    Cntx Swtch Out    89.9719
==========================================================================
89.9719    [I A I I ]  EMPTY    Cntx Swtch In     90.0719
==========================================================================
90.0719    [I A I I ]  EMPTY    Departure         90.953
==========================================================================
90.953     [I I I I ]  EMPTY    Arrival           91.2877
==========================================================================
91.2877    [A I I I ]  EMPTY    Cntx Swtch In     91.3877
==========================================================================
91.3877    [A I I I ]  EMPTY    Arrival           91.4158
==========================================================================
91.4158    [A A I I ]  EMPTY    Cntx Swtch In     91.5158
==========================================================================
```

## EventHandler ManageEvent

```
void EventHandler::manageEvent(Event event){

    switch (event.type)
    {
     case ARRIVAL:
         this->arrive(event);
         break;

     case DEPARTURE:
         this->depart(event);
         break;

     case CONTEXTSWITCHIN:
         contextSwitchIn(event);
         break;

     case CONTEXTSWITCHOUT:
         contextSwitchOut(event);
         break;

     default:
         break;
    }
}
```

# Simulation Experiments and Results

# Experiments Performed

- The simulation was run multiple times for random values of service times, timeout times, and think times with the same mean for the same number of users.
- The mean of all the runs is considered.
- The above process was repeated for different number of users.
- Response times, CPU utilization, throughput, and request drops are plotted.
- Confidence interval is also plotted for response time to get a clear idea.

# Comparison With Measurement Data

# System Configuration :

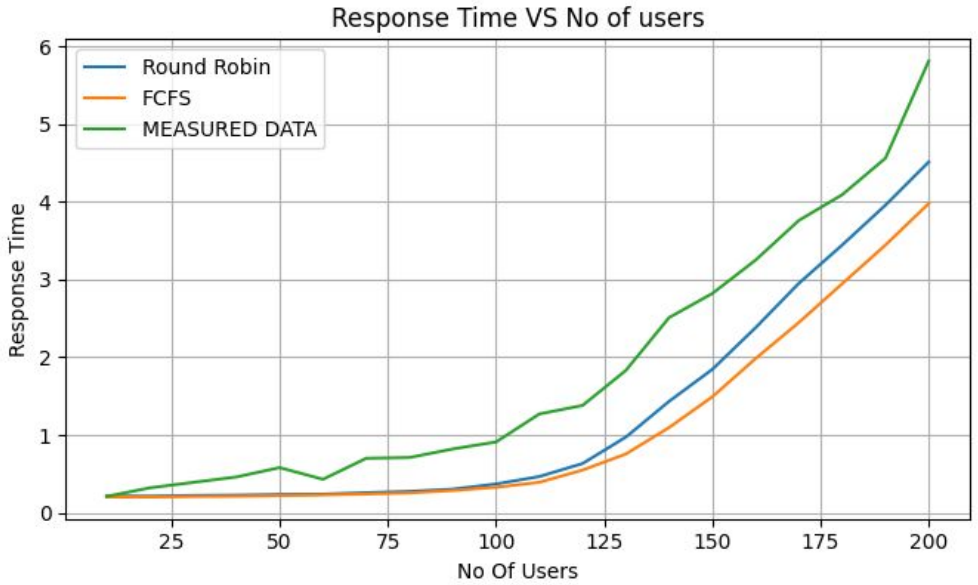
1. Number of Cores: 4
2. Number of Threads per Core: 1
3. Mean Service Time: Exponential (Mean: 0.2 sec)
4. Mean Timeout Time: 50 sec + Exponential (Mean : 5 sec)
5. Context Switch Time (Only for Round-Robin): 0.1 sec
6. Time Quantum (Only for Round-Robin): 0.5 sec

# Experiments Performed

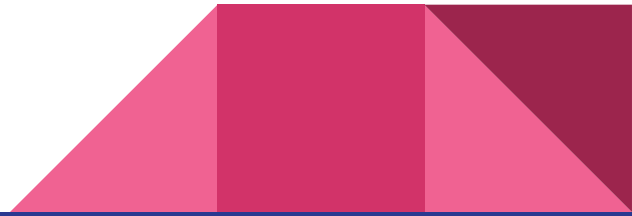
- To compare the simulation outputs with the real system, we plotted values obtained from measurement analysis of the apache server and our simulation, for the same configurations.
- Response time, throughput, and CPU utilization were compared.
- All the metrics showed great similarity in the apache server and our simulation.

# Response Time Vs Number Of Users

- The graphs of measured values and simulation values show very similar trends.
- Although the response time with the round-robin scheduling policy is more than the FCFS policy for all user values.
- This happens because of the context switching in the round-robin policy.
- The saturation number can be found using the response time graph
- **M* = c + c*(1/service time)*think time**
- **M* = 4 + 4*(1/0.2)*6 = 4 + 4*5*6 = 124**

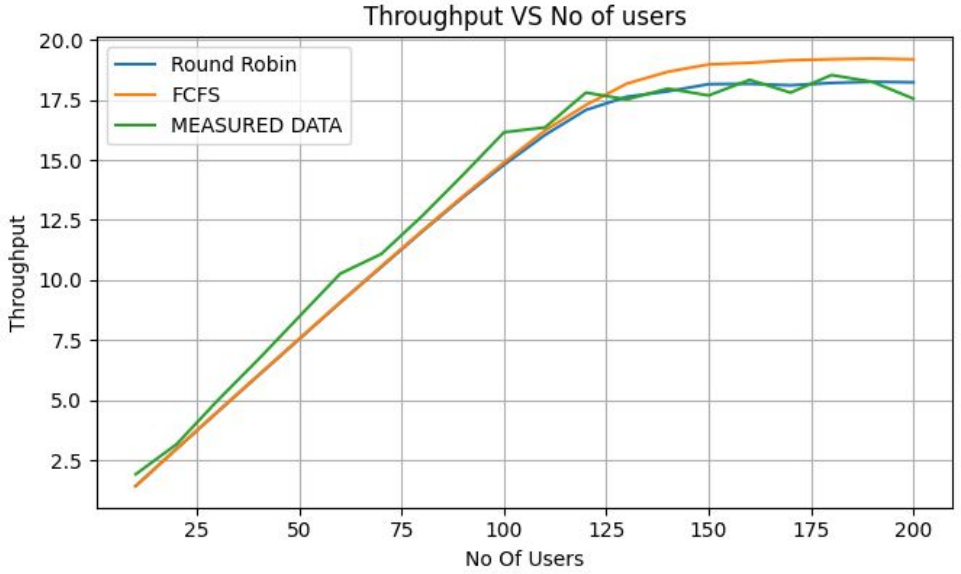


- From graphs, it is clear that the system saturates near 120 users.
- The response time of the measured value is a bit higher than the simulation values. The reason behind this is, in practical systems, there are many more factors affecting the response time that we haven’t modeled in the simulation.



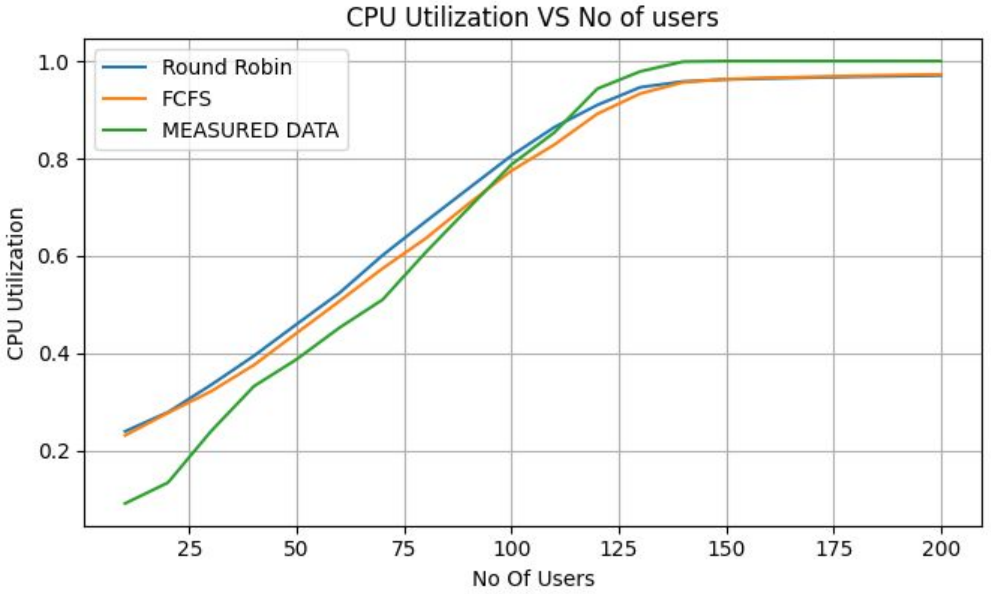
# Throughput Vs Number of Users

- The graphs of measured and simulation values show similar trends.
- The throughput increases initially and saturates at a value of 18 req/sec for round robin and measured values.
- For FCFS system throughput reaches 19 req/sec.
- The system saturates around 125 users.



# CPU Utilization Vs Number of Users

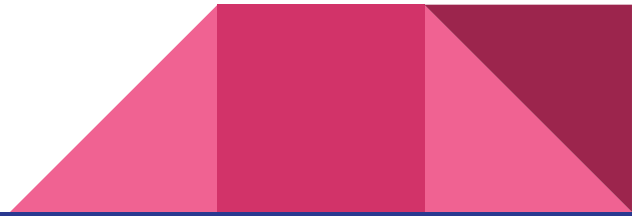
- The graphs of measured and simulation values show similar trends.
- The utilization reaches the maximum value of 1, around 125 users.
- As context switch time is 0 in this case, even in the case of the round-robin, utilization reaches up to 100%.



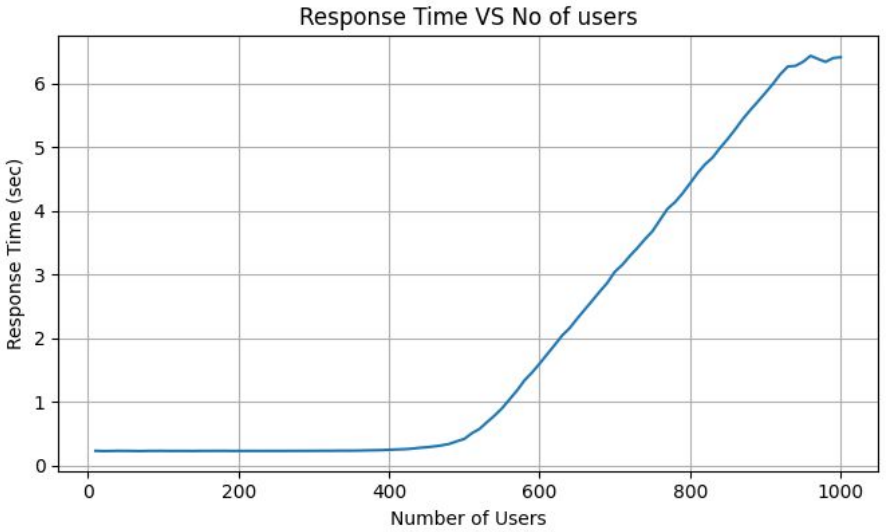
# Multithreaded Web Server Simulation

# System Configuration

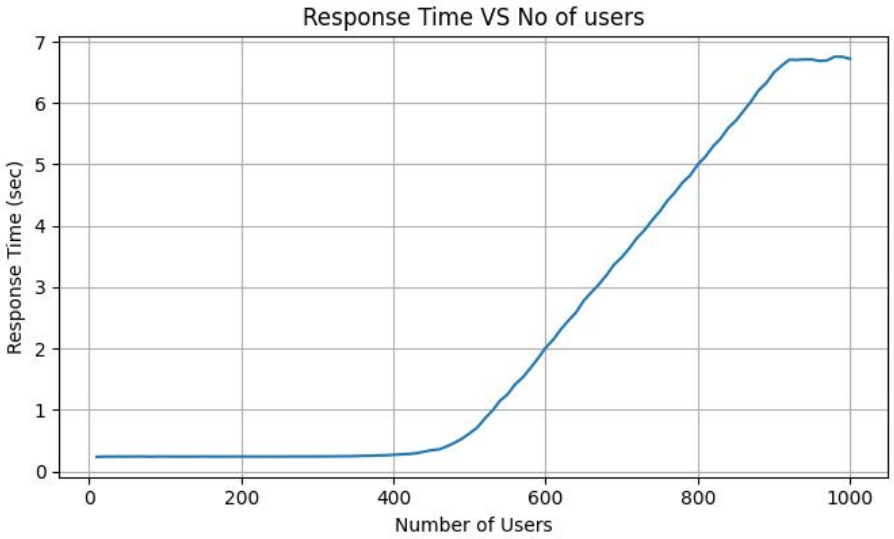
1. Number of Cores: 4
2. Number of Threads per Core: 4
3. Mean Service Time: Exponential (Mean: 0.25 sec)
4. Mean Timeout Time: 50 sec + Exponential (Mean: 5sec)
5. Context Switch Time (Only for Round-Robin): 0.01sec
6. Time Quantum (Only for Round-Robin): 0.5 sec



# Response Time Vs Number of Users

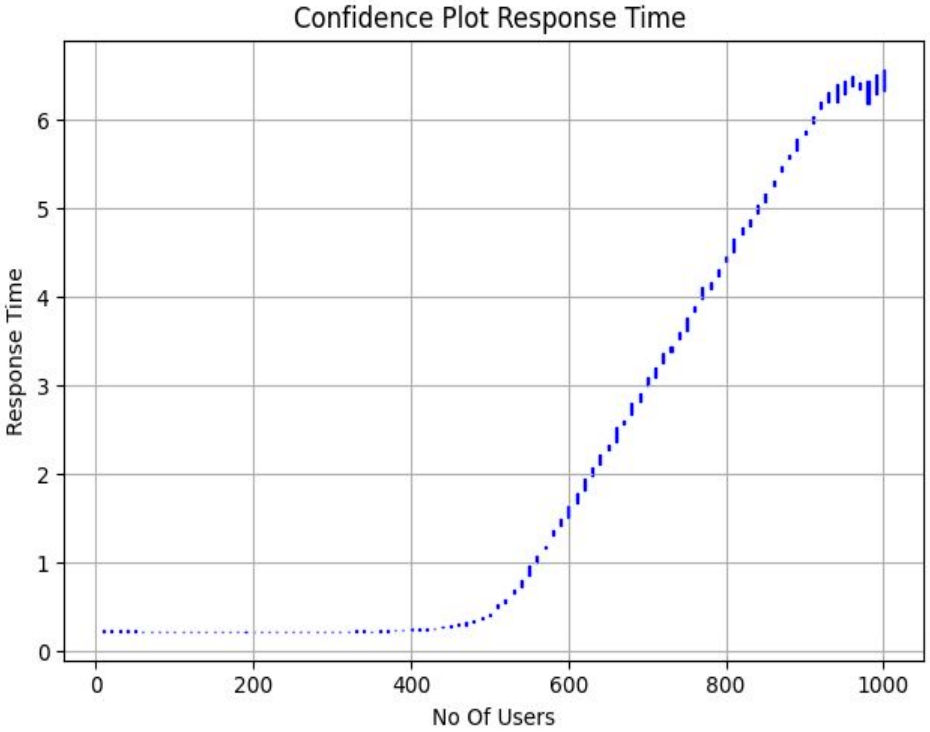



FCFS

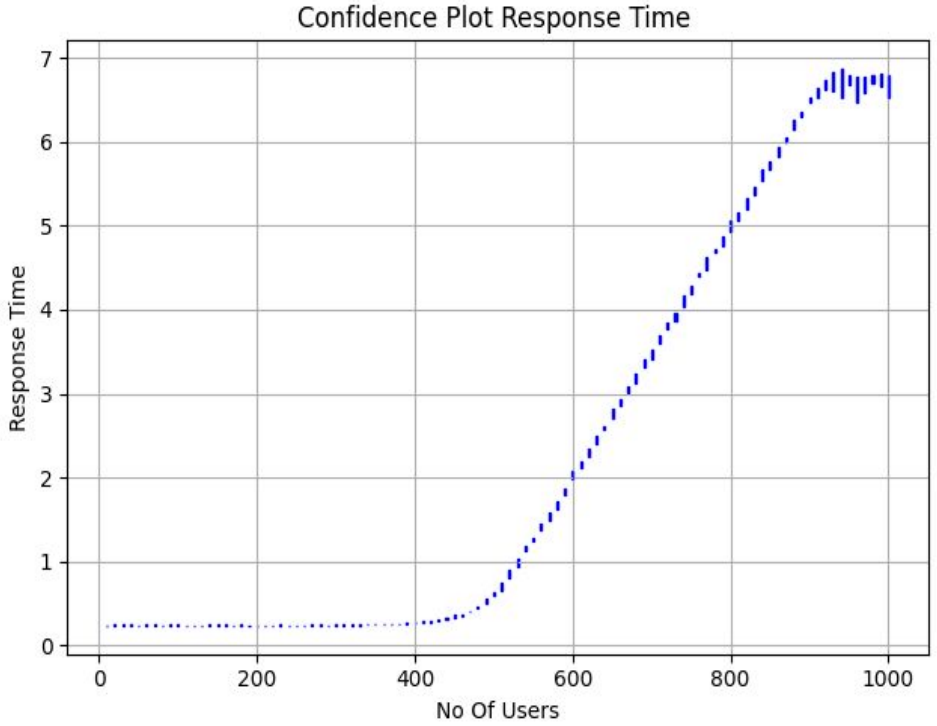



Round Robin

# Confidence Interval Graph for Response Time

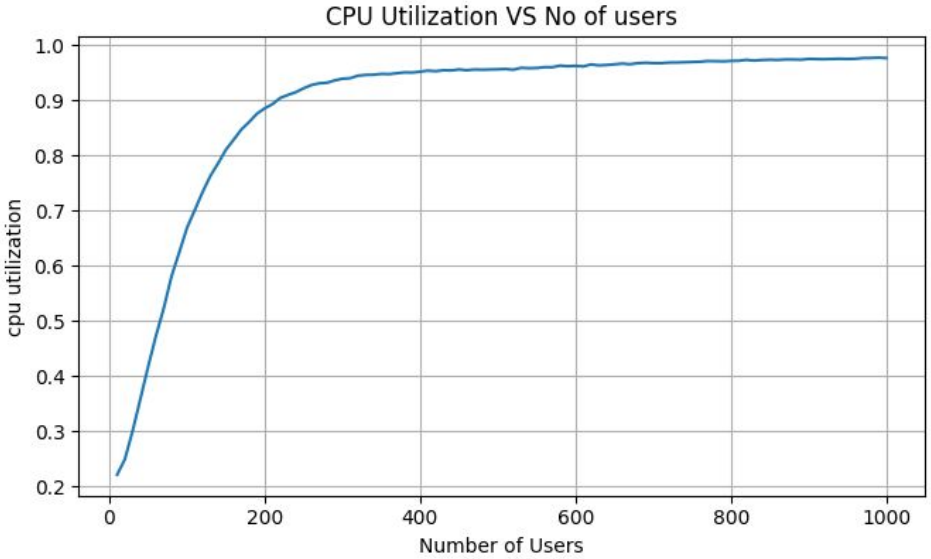



FCFS

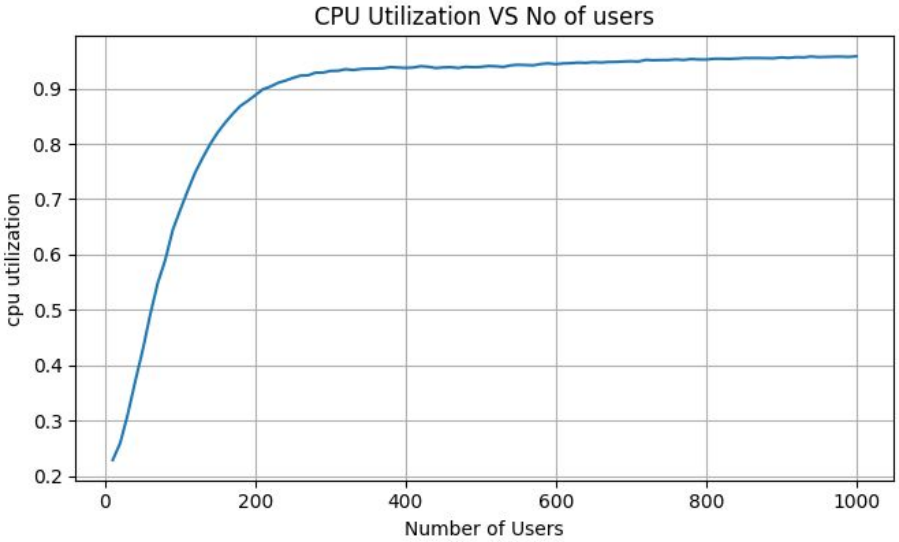



Round Robin

# CPU Utilization Vs Number Of Users

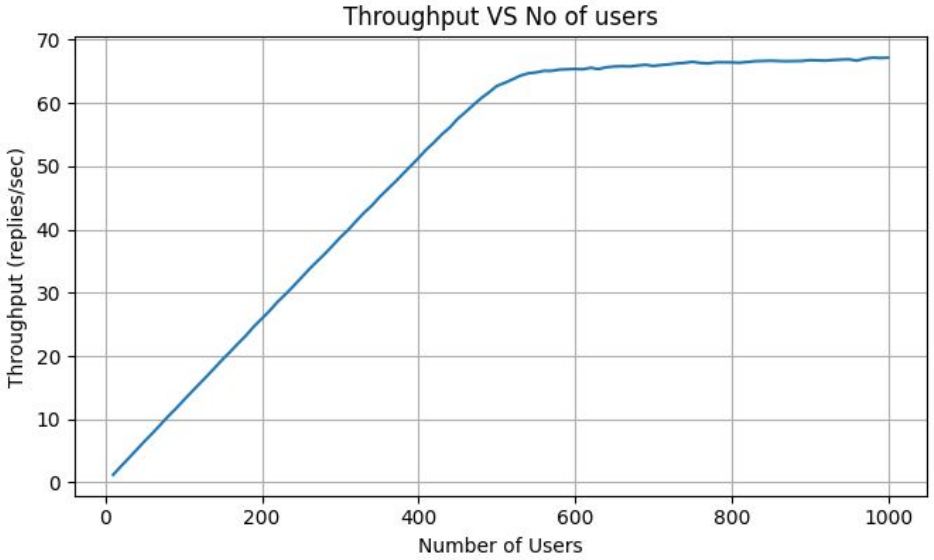



FCFS

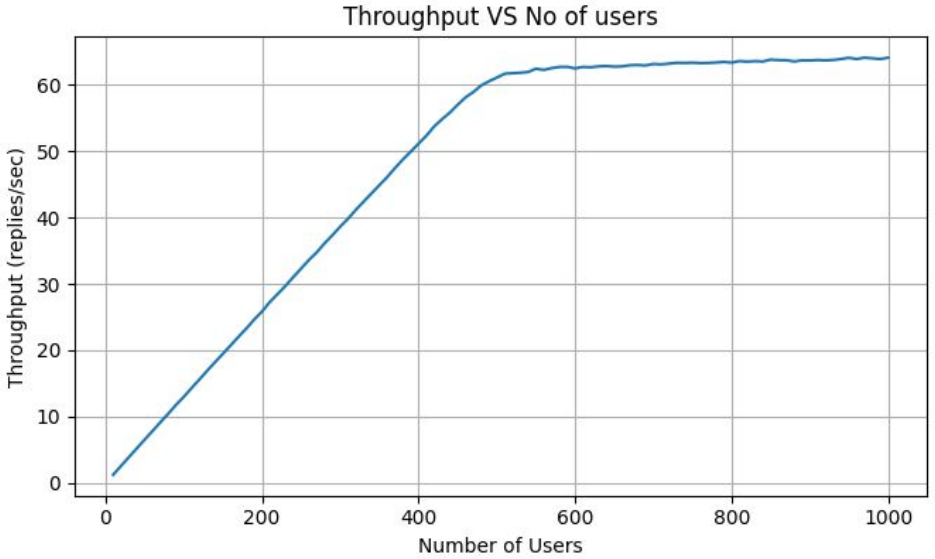



Round Robin

# Throughput Vs Number Of Users

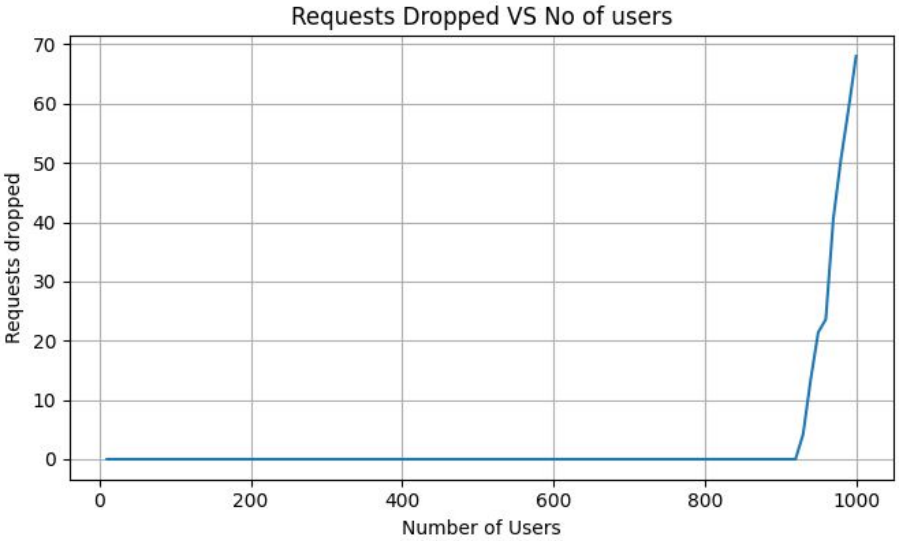



FCFS

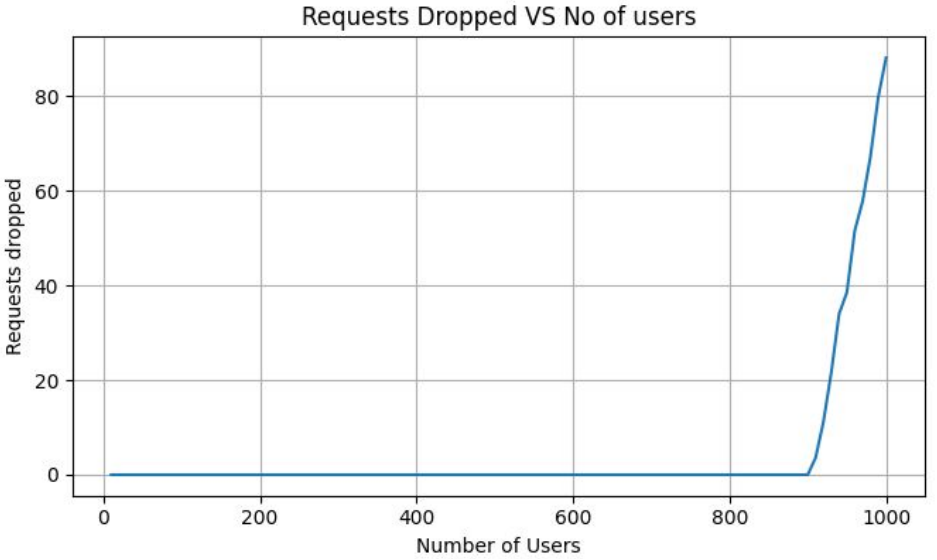



Round Robin

# Requests Drops Vs Number Of Users




FCFS




Round Robin

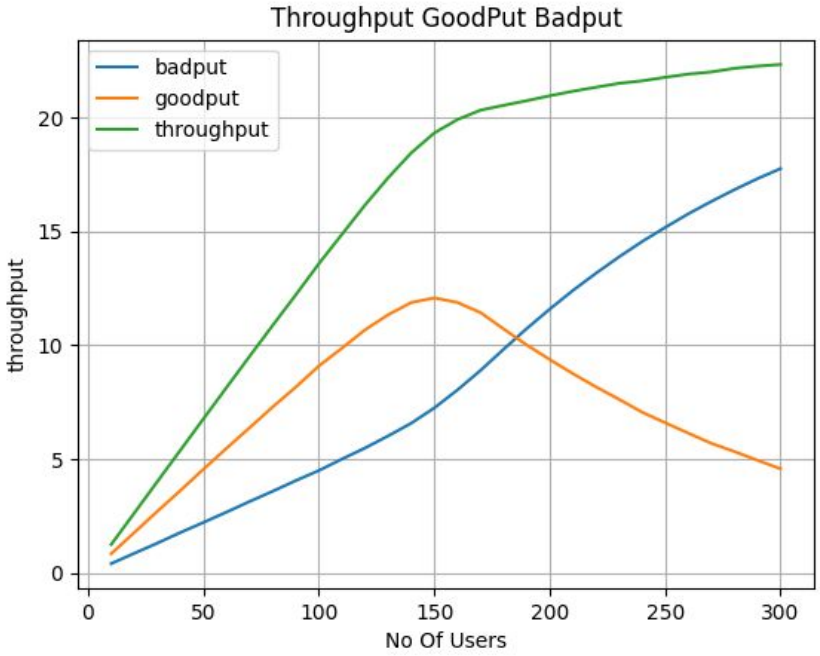
# Curiosity Experiment 1 : Decreased Timeout Value

# System Configuration :

1. Number of Cores: 4
2. Number of Threads per Core: 1
3. Mean Service Time: Exponential (Mean: 0.25 sec)
4. Mean Timeout Time: 5 sec + Exponential (Mean : 5 sec)
5. Context Switch Time (Only for Round-Robin): 0.01 sec
6. Time Quantum (Only for Round-Robin): 0.5 sec

# Throughput, Goodput, Bad-put Comparison

- Experiments were conducted to check the effect of timeout time.
- The graph highlights the effects on throughput, goodput, and bad-put when minimum timeout time is reduced.
- With the decrease in timeout time, bad-put increased after a certain number of users.
- This happens because more and more requests timeout, increasing retries.



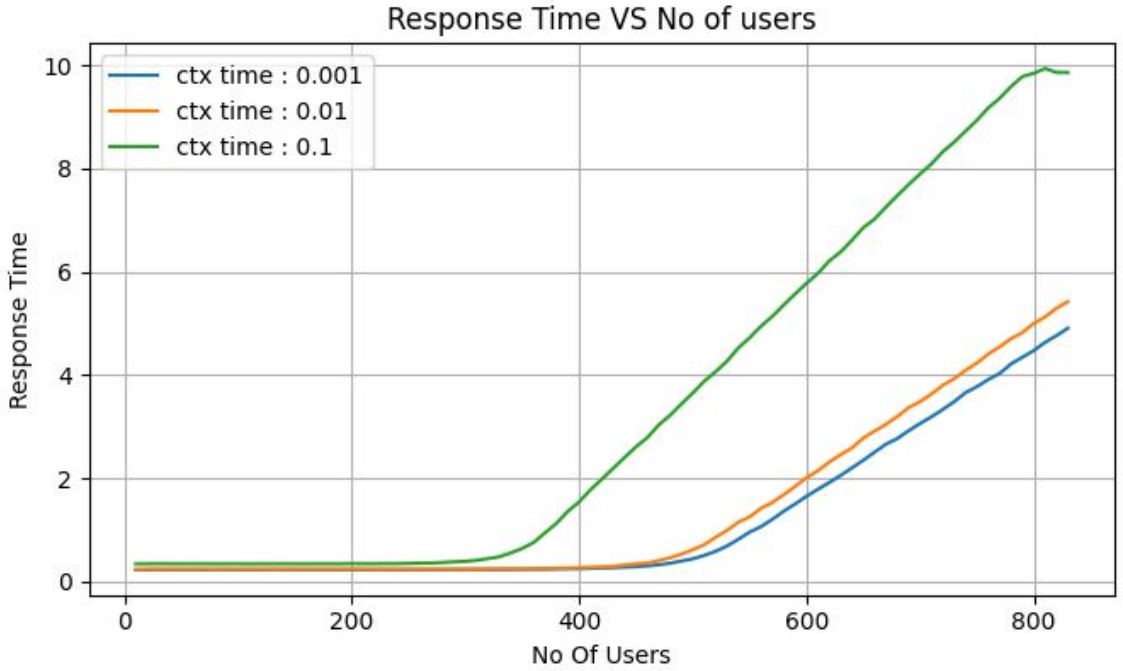
# Curiosity Experiment 2 : Context Switch Time Variation

# System Configuration :

1. Number of Cores: 4
2. Number of Threads per Core: 4
3. Mean Service Time: Exponential (Mean: 0.25 sec)
4. Mean Timeout Time: 50 sec + Exponential (Mean : 5 sec)
5. Context Switch Time (Only for Round-Robin): 0.001sec, 0.01 sec , 0.1sec
6. Time Quantum (Only for Round-Robin): 0.5 sec

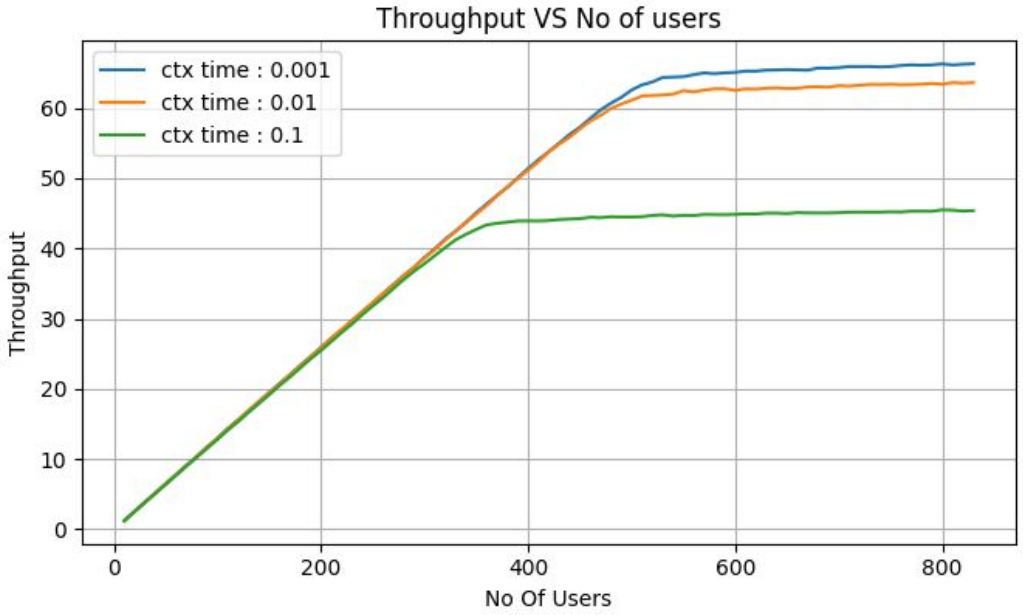
# Response time

- The graph shows the effect of context switch time on response times.
- As the context switch time increases, the response time also increases.
- The context switch is an overhead for the server. The increase in context switch time increases the overhead for each request, increasing the response time.



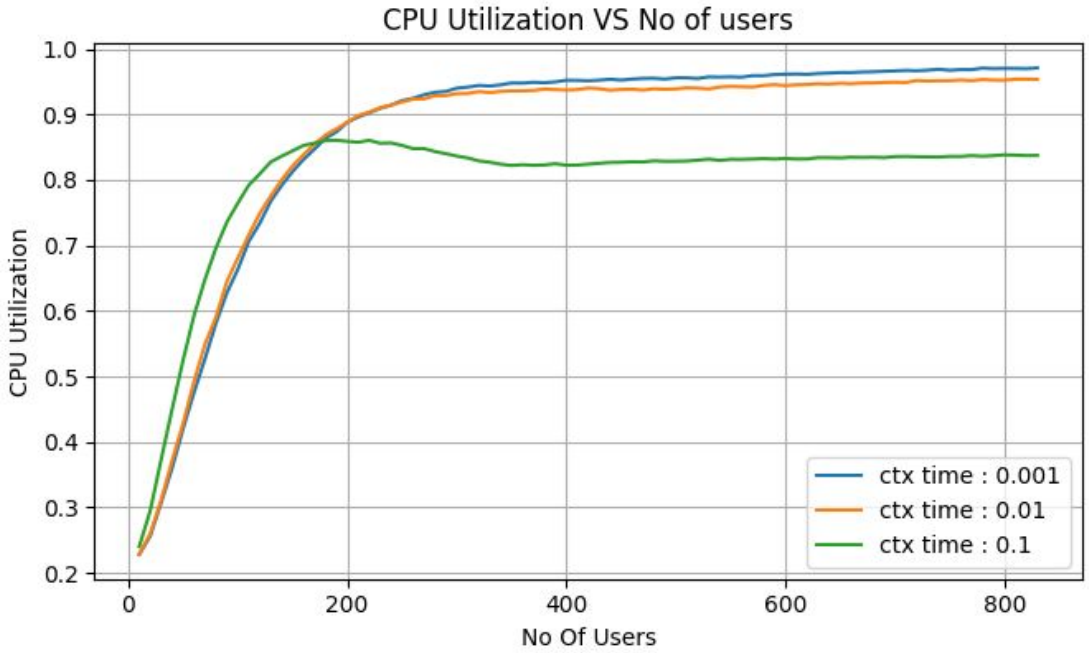
# Throughput

- Contrary to response time, throughput decreases when context switch time increases.
- The reason is the same. Context switching is an overhead for the server, so the time spent in context switching is not a useful time. So, because of more context switching time, fewer overall requests are processed in the same amount of time resulting in less throughput.



# Utilization

- Similar to throughput, utilization also decreases with an increase in context switch time.
- As we don’t consider the context switching time to be useful, we consider the CPU to be idle for that time.
- Utilization is defined as the fraction of time the CPU is busy.
- As context switching time increases idle time of CPU. More context switch time decreases utilization.



# Conclusion

- We implemented a web server simulation program and analysed it using the metrics like throughput, response time, CPU utilization and request drops.
- We also compared the performance with measurements we got from apache server analysis. The comparison showed great similarity in both the systems.
- We also performed some experiments to check the effect of timeouts and context switch times on performance of the web server.
- On decreasing the minimum timeout value, the bad-put increased after some number of users.
- On increasing the context switch time, response time increased while the throughput and CPU utilisation decreased.